## Introduction

Machine Learning techniques are broadly divided into two parts:

1. Supervised Machine Learning
2. Unsupervised Machine Learning

In Supervised Machine Learning, the data is labelled and the algorithm learns from labelled training data. Examples of this method are Classification and Regression.

In Unsupervised Machine Learning, we do not need to supervise the model. Such a method deals with unlabelled data. Unsupervised machine learning helps us find hidden and unknown patterns in data.

Often it easier to get unlabelled data as compared to labelled data, and in such cases, we can use unsupervised machine learning to work on the data. Data, which needs categorization can be categorized with the help of unsupervised machine learning.

Clustering is a type of unsupervised machine learning in which the algorithm processes our data and divided them into “clusters”.

## The Clustering Explained

Clustering algorithms try to find natural clusters in data, the various aspects of how the algorithms to cluster data can be tuned and modified. Clustering is based on the principle that items within the same cluster must be similar to each other. The data is grouped in such a way that related elements are close to each other.

Diverse and different types of data are subdivided into smaller groups.

## Uses of Clustering

### Marketing:

In the field of marketing, clustering can be used to identify various customer groups with existing customer data. Based on that, customers can be provided with discounts, offers, promo codes etc.

### Real Estate:

Clustering can be used to understand and divide various property locations based on value and importance. Clustering algorithms can process through the data and identify various groups of property on the basis of probable price.

### BookStore and Library management:

Libraries and Bookstores can use Clustering to better manage the book database. With proper book ordering, better operations can be implemented.

### Document Analysis:

Often, we need to group together various research texts and documents according to similarity. And in such cases, we don’t have any labels. Manually labelling large amounts of data is also not possible. Using clustering, the algorithm can process the text and group it into different themes.

These are some of the interesting use cases of clustering.

## K-Means Clustering

K-Means clustering is an unsupervised machine learning algorithm that divides the given data into the given number of clusters. Here, the “K” is the given number of predefined clusters, that need to be created.

It is a centroid based algorithm in which each cluster is associated with a centroid. The main idea is to reduce the distance between the data points and their respective cluster centroid.

The algorithm takes raw unlabelled data as an input and divides the dataset into clusters and the process is repeated until the best clusters are found.

K-Means is very easy and simple to implement. It is highly scalable, can be applied to both small and large datasets. There is, however, a problem with choosing the number of clusters or K. Also, with the increase in dimensions, stability decreases. But, overall K Means is a simple and robust algorithm that makes clustering very easy.

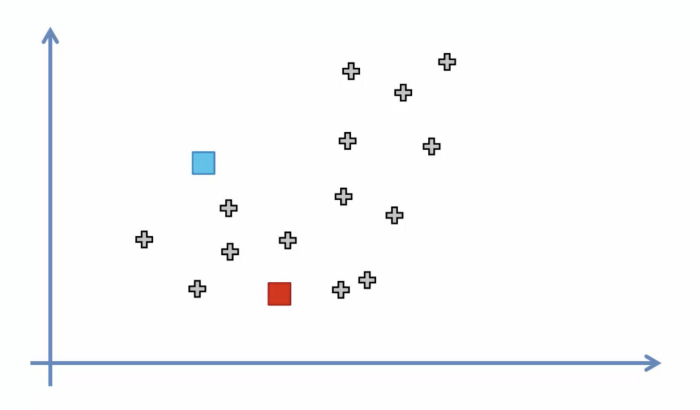
K-Means Clustering is an unsupervised machine learning algorithm. In contrast to traditional supervised machine learning algorithms, K-Means attempts to classify data without having first been trained with labeled data. Once the algorithm has been run and the groups are defined, any new data can be easily assigned to the most relevant group.

The real world applications of K-Means include:

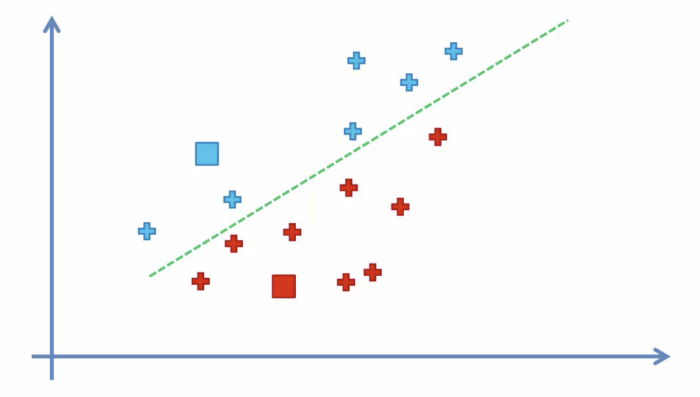
* customer profiling
* market segmentation
* computer vision
* search engines
* astronomy

# **How it works**

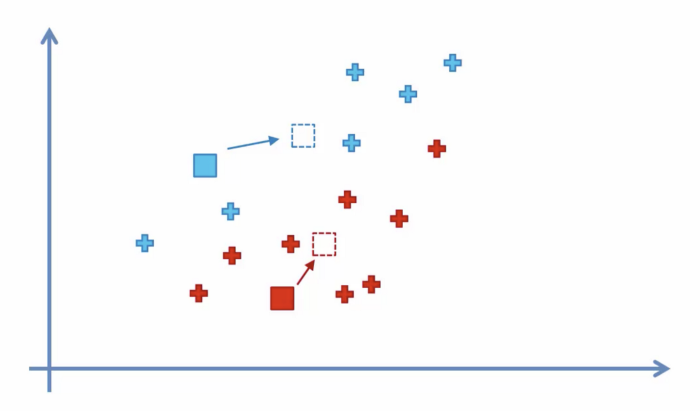
1. Select **K**(i.e. 2)random points as cluster centers called centroids



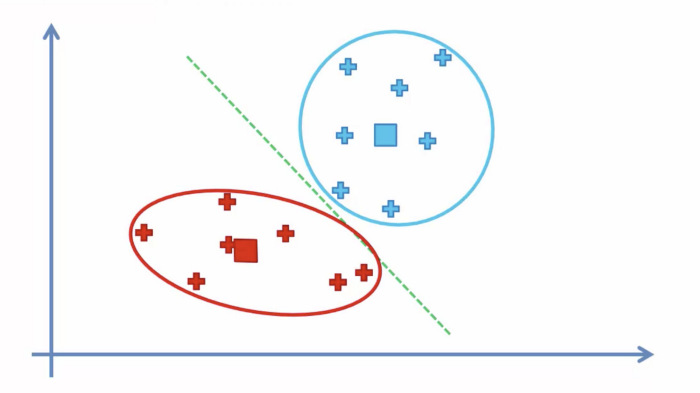
2. Assign each data point to the closest cluster by calculating its distance with respect to each centroid



3. Determine the new cluster center by computing the average of the assigned points



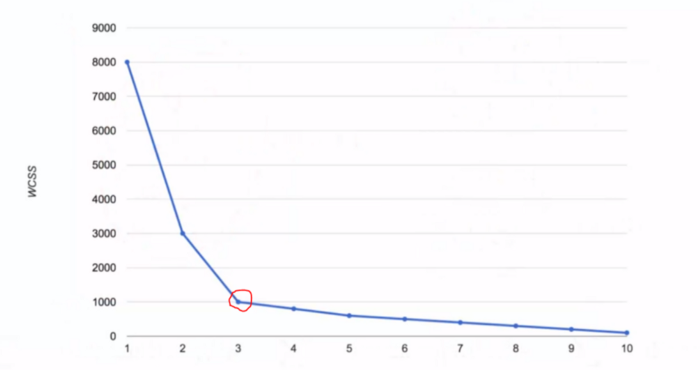
4. Repeat steps 2 and 3 until none of the cluster assignments change



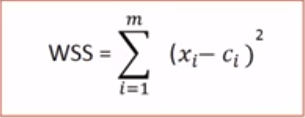
# **Choosing the right number of clusters**

Often times the data you’ll be working with will have multiple dimensions making it difficult to visual. As a consequence, the optimum number of clusters is no longer obvious. Fortunately, we have a way of determining this mathematically.

We graph the relationship between the number of clusters and Within Cluster Sum of Squares (WCSS) then we select the number of clusters where the change in WCSS begins to level off (elbow method).



WCSS is defined as the sum of the squared distance between each member of the cluster and its centroid.



Context

This data set is created only for the learning purpose of the customer segmentation concepts, also known as market basket analysis.

### Content

You are owing a supermarket mall and through membership cards, you have some basic data about your customers like Customer ID, age, gender, annual income and spending score.  
Spending Score is something you assign to the customer based on your defined parameters like customer behavior and purchasing data.

**Problem Statement**  
You own the mall and want to understand the customers like who can be easily converge [Target Customers] so that the sense can be given to marketing team and plan the strategy accordingly.

### Inspiration

By the end of this case study , you would be able to answer below questions.  
1- How to achieve customer segmentation using machine learning algorithm (KMeans Clustering) in Python in simplest way.  
2- Who are your target customers with whom you can start marketing strategy [easy to converse]  
3- How the marketing strategy works in real world

The data includes the following features:

1. Customer ID

2. Customer Gender

3. Customer Age

4. Annual Income of the customer (in Thousand Dollars)

5. Spending score of the customer (based on customer behaviour and spending nature)

Let us proceed with the code.

#Importing the necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from mpl\_toolkits.mplot3d import Axes3D

%matplotlib inline

#%matplotlib inline sets the backend of matplotlib to the 'inline' backend: With this #backend, the output of plotting commands is displayed inline within frontends like #the Jupyter notebook, directly below the code cell that produced it. The resulting #plots will then also be stored in the notebook document.

The necessary libraries are imported.

#Reading the excel file

data=pd.read\_csv("Mall\_Customers.csv")

The data is read.

The data has 200 entries, that is data from 200 customers.

data.head()

So let us have a look at the data.

data.corr()

#corr() is used to find the pairwise correlation of all columns in the dataframe. Any #na values are automatically excluded. For any non-numeric data type columns in the #dataframe it is ignored.

#Note: The correlation of a variable with itself is 1.

#A correlation coefficient is used in statistics to describe a pattern or #relationship between two variables. A negative correlation describes the extent to #which two variables move in opposite directions. For example, for two variables, X and Y, an increase in X is associated with a decrease in Y

The data seems to be interesting. Let us look at the data distribution.

**Annual Income Distribution:**

#Distribution of Annnual Income

plt.figure(figsize=(10, 6))

sns.set(style = 'whitegrid')

sns.distplot(data['Annual Income (k$)'])

plt.title('Distribution of Annual Income (k$)', fontsize = 20)

plt.xlabel('Range of Annual Income (k$)')

plt.ylabel('Count')

#The seaborn. distplot() function is used to plot the distplot. The distplot #represents the univariate distribution of data i.e. data distribution of a variable #against the density distribution.

Most of the annual income falls between 50K to 85K.

**Age Distribution:**

#Distribution of age

plt.figure(figsize=(10, 6))

sns.set(style = 'whitegrid')

sns.distplot(data['Age'])

plt.title('Distribution of Age', fontsize = 20)

plt.xlabel('Range of Age')

plt.ylabel('Count')

There are customers of a wide variety of ages.

**Spending Score Distribution:**

#Distribution of spending score

plt.figure(figsize=(10, 6))

sns.set(style = 'whitegrid')

sns.distplot(data['Spending Score (1-100)'])

plt.title('Distribution of Spending Score (1-100)', fontsize = 20)

plt.xlabel('Range of Spending Score (1-100)')

plt.ylabel('Count')

The maximum spending score is in the range of 40 to 60.

### Gender Analysis:

genders = data.Gender.value\_counts()

sns.set\_style("darkgrid")

plt.figure(figsize=(10,4))

sns.barplot(x=genders.index, y=genders.values)

plt.show()

More female customers than male.

### Clustering based on 2 features

First, we work with two features only, annual income and spending score.

#We take just the Annual Income and Spending score

df1=data[["CustomerID","Gender","Age","Annual Income (k$)","Spending Score (1-100)"]]

X=df1[["Annual Income (k$)","Spending Score (1-100)"]]

#The input data

X.head()

#Scatterplot of the input data

plt.figure(figsize=(10,6))

sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)', data = X ,s = 60 )

#s - size in points^2. It is a scalar or an array of the same length as x and y.

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Spending Score (1-100) vs Annual Income (k$)')

plt.show()

The data does seem to hold some patterns.

#Importing KMeans from sklearn

from sklearn.cluster import KMeans

Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k. Next, we choose the k for which WSS first starts to diminish. This value of K gives us the best number of clusters to make from the raw data.

wcss=[]

for i in range(1,11):

km=KMeans(n\_clusters=i)

#n\_clusters - The number of clusters to form as well as the number of centroids to #generate

km.fit(X)

wcss.append(km.inertia\_)

#inertia\_ -Sum of squared distances of samples to their closest cluster center

#The elbow curve

plt.figure(figsize=(12,6))

plt.plot(range(1,11),wcss)

plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")

plt.xlabel("K Value")

plt.xticks(np.arange(1,11,1))

plt.ylabel("WCSS")

plt.show()

The plot:

This is known as the elbow graph, the x-axis being the number of clusters, the number of clusters is taken at the elbow joint point. This point is the point where making clusters is most relevant as here the value of WCSS suddenly stops decreasing. Here in the graph, after 5 the drop is minimal, so we take 5 to be the number of clusters.

#Taking 5 clusters

km1=KMeans(n\_clusters=5)

#Fitting the input data

km1.fit(X)

#predicting the labels of the input data

y=km1.predict(X)

#adding the labels to a column named label

df1["label"] = y

#The new dataframe with the clustering done

df1.head()

The labels added to the data.

#Scatterplot of the clusters

plt.figure(figsize=(10,6))

sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)',hue="label",

palette=['green','orange','brown','dodgerblue','red'], legend='full',data = df1 ,s = 60 )

#We use the combination of hue and pallete to color the data points in scatter plot

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.title('Spending Score (1-100) vs Annual Income (k$)')

plt.show()

We can clearly see that 5 different clusters have been formed from the data. The red cluster is the customers with the least income and least spending score, similarly, the blue cluster is the customers with the most income and most spending score.

### k-Means Clustering on the basis of 3D data

Now, we shall be working on 3 types of data. Apart from the spending score and annual income of customers, we shall also take in the age of the customers.

#Taking the features

df2=data[["CustomerID","Gender","Age","Annual Income (k$)","Spending Score (1-100)"]]

X2=df2[["Age","Annual Income (k$)","Spending Score (1-100)"]]

#Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k.

wcss = []

for k in range(1,11):

kmeans = KMeans(n\_clusters=k, init="k-means++")

#k-means++ - selects initial cluster centers for k-mean clustering in a smart way to #speed up convergence

kmeans.fit(X2)

wcss.append(kmeans.inertia\_)

plt.figure(figsize=(12,6))

plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")

#Marker is a special way of handling markers in graphs.

plt.xlabel("K Value")

plt.xticks(np.arange(1,11,1))

plt.ylabel("WCSS")

plt.show()

The WCSS curve.

Here can assume that K=5 will be a good value.

#We choose the k for which WSS starts to diminish

km2 = KMeans(n\_clusters=5)

y2 = km.fit\_predict(X2)

df2["label"] = y2

#The data with labels

df2.head()

The data:

Now we plot it.

#3D Plot as we did the clustering on the basis of 3 input features

fig = plt.figure(figsize=(20,10))

ax = fig.add\_subplot(111, projection='3d')

#subplot() function adds subplot to a current figure at the specified grid position

ax.scatter(df2.Age[df2.label == 0], df2["Annual Income (k$)"][df2.label == 0], df2["Spending Score (1-100)"][df2.label == 0], c='purple', s=60)

ax.scatter(df2.Age[df2.label == 1], df2["Annual Income (k$)"][df2.label == 1], df2["Spending Score (1-100)"][df2.label == 1], c='red', s=60)

ax.scatter(df2.Age[df2.label == 2], df2["Annual Income (k$)"][df2.label == 2], df2["Spending Score (1-100)"][df2.label == 2], c='blue', s=60)

ax.scatter(df2.Age[df2.label == 3], df2["Annual Income (k$)"][df2.label == 3], df2["Spending Score (1-100)"][df2.label == 3], c='green', s=60)

ax.scatter(df2.Age[df2.label == 4], df2["Annual Income (k$)"][df2.label == 4], df2["Spending Score (1-100)"][df2.label == 4], c='yellow', s=60)

ax.view\_init(35, 185)

#Use view\_init() can be used to rotate the axes programmatically

plt.xlabel("Age")

plt.ylabel("Annual Income (k$)")

ax.set\_zlabel('Spending Score (1-100)')

plt.show()

The output:

What we get is a 3D plot. Now, if we want to know the customer IDs, we can do that too.

cust1=df2[df2["label"]==1]

print('Number of customer in 1st group=', len(cust1))

print('They are -', cust1["CustomerID"].values)

print("--------------------------------------------")

cust2=df2[df2["label"]==2]

print('Number of customer in 2nd group=', len(cust2))

print('They are -', cust2["CustomerID"].values)

print("--------------------------------------------")

cust3=df2[df2["label"]==0]

print('Number of customer in 3rd group=', len(cust3))

print('They are -', cust3["CustomerID"].values)

print("--------------------------------------------")

cust4=df2[df2["label"]==3]

print('Number of customer in 4th group=', len(cust4))

print('They are -', cust4["CustomerID"].values)

print("--------------------------------------------")

cust5=df2[df2["label"]==4]

print('Number of customer in 5th group=', len(cust5))

print('They are -', cust5["CustomerID"].values)

print("--------------------------------------------")

The output we get:

Number of customer in 1st group= 24

They are - [129 131 135 137 139 141 145 147 149 151 153 155 157 159 161 163 165 167

169 171 173 175 177 179]

——————————————–

Number of the customer in 2nd group= 29

They are - [ 47 51 55 56 57 60 67 72 77 78 80 82 84 86 90 93 94 97

99 102 105 108 113 118 119 120 122 123 127]

——————————————–

Number of the customer in 3rd group= 28

They are - [124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158

160 162 164 166 168 170 172 174 176 178]

——————————————–

Number of the customer in 4th group= 22

They are - [ 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 46]

--------------------------------------------

Number of customer in 5th group= 12

They are - [ 3 7 9 11 13 15 23 25 31 33 35 37]

——————————————–

So, we used K-Means clustering to understand customer data. K-Means is a good clustering algorithm. Almost all the clusters have similar density. It is also fast and efficient in terms of computational cost.